Making it possible to build safe and secure systems with Linux

Federico Arrighetti - Elektrobit Automotive GmbH

Michael Armbruster - emlix GmbH

21 May 2025

What ever happened to Baby Safety?

Safety-critical and mission-critical systems have grown up:

- functionalities have become more and more complex
- …and more and more diverse too
- the amount of data to compute has dramatically increased
- systems on chip are not the microprocessors they used to be
- …not to mention sensors, actuators and other more advanced devices to interface
- the need for the co-existence of applications having different levels of criticality is growing
- cyber-security has joined the party
- and of course pressure on development costs has never left the playground

The answer, my friend, is blowin' in the Linux

Linux does not lack virtues:

- open-source means rapid evolution cycle, continuous maintenance, and quick bug-fixing
- no vendor lock-in
- supports most hardware
- cyber-security is built-in
- fully scalable for the specific needs of each project
- plenty of skilled developers available on the market
- extremely broad use in the most diverse and demanding contexts provides an absolutely non-episodical evidence of its reliability

Linux would be the answer, if it just weren't for a detail: **IT IS NOT DEPENDABLE!**

But then, let's make Linux dependable

It's not rocket science, it just takes to follow the norms: produce the design specification (requirements, architecture…) and then test accordingly.

Well, maybe it's not rocket science, but Linux is big and complex, very big and very complex:

- functional specification is sparse, incomplete, and of questionable quality
- Linux is monolithic, difficult to decompose for detailed specification and testing
- automatic analysis tools may show the complexity, but they do not make it less complex
- and all that needs to be repeated for each new release (so, quite often)

Smells of "reverse engineering" from a mile.
Maybe just a bit better mannered or sugar-coated, but still "reverse" engineering.
Or maybe post-mortem examination…

If you don’t like the solution, change the problem

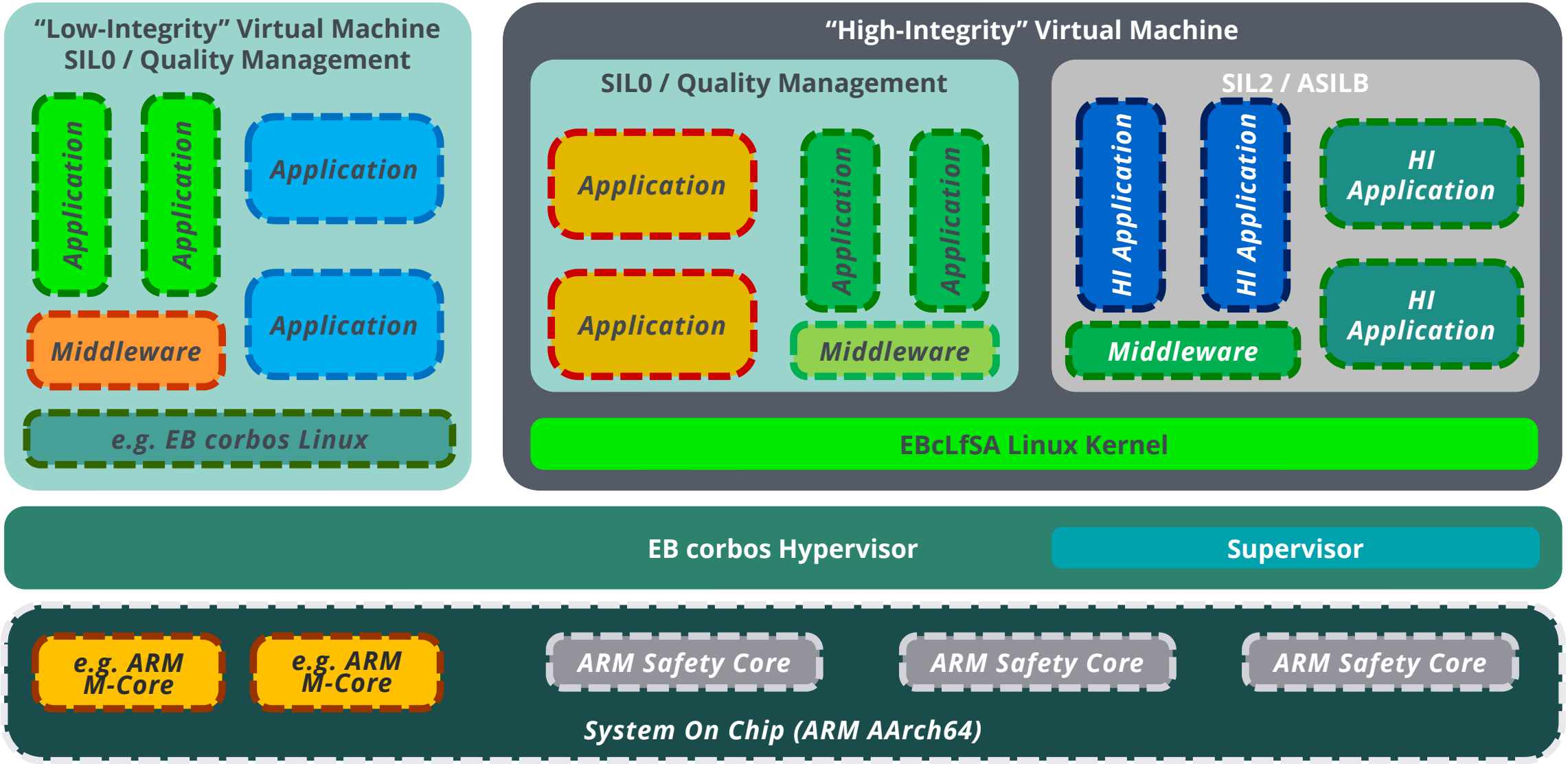
We at Elektrobit with emlix believe that Linux **IS** dependable, just that we cannot demonstrate that it is, and to be honest we are not that keen on trying.

Instead, we just detect when Linux is NOT dependable:

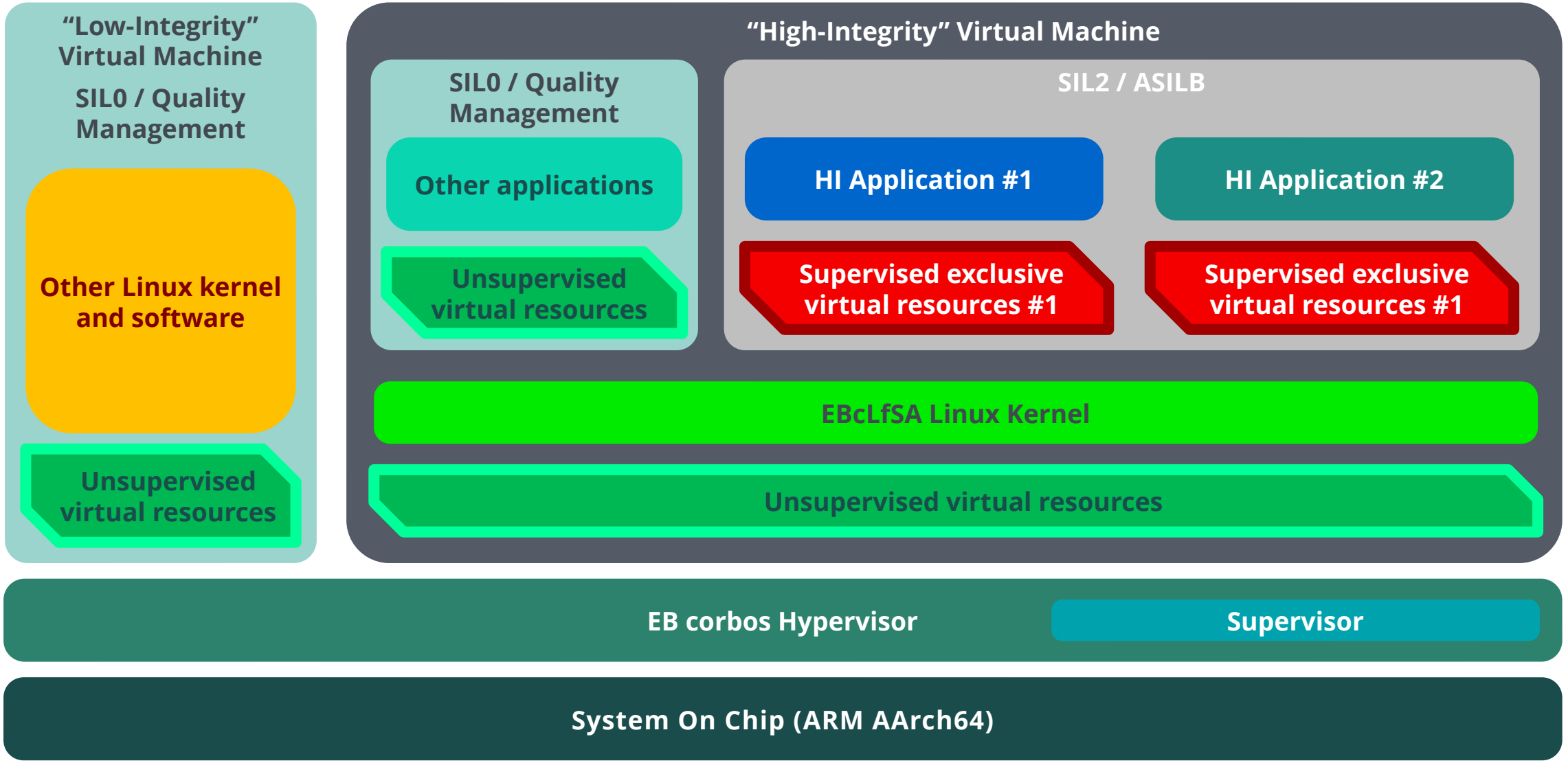
- we are not interested in Linux itself
- we do not need an assessment of Linux or of any software
- we are interested in the cyber-physical systems built on Linux
- we focus on the application software and not on the operating system
- we add to Linux a “guardian angel” who detects when Linux misbehaves
- we limit access rights of Linux using the features of the ARM architecture

And not only it works, but it has been positively assessed by TÜV Nord for SIL2 EN 61508 / ASILB ISO 26262

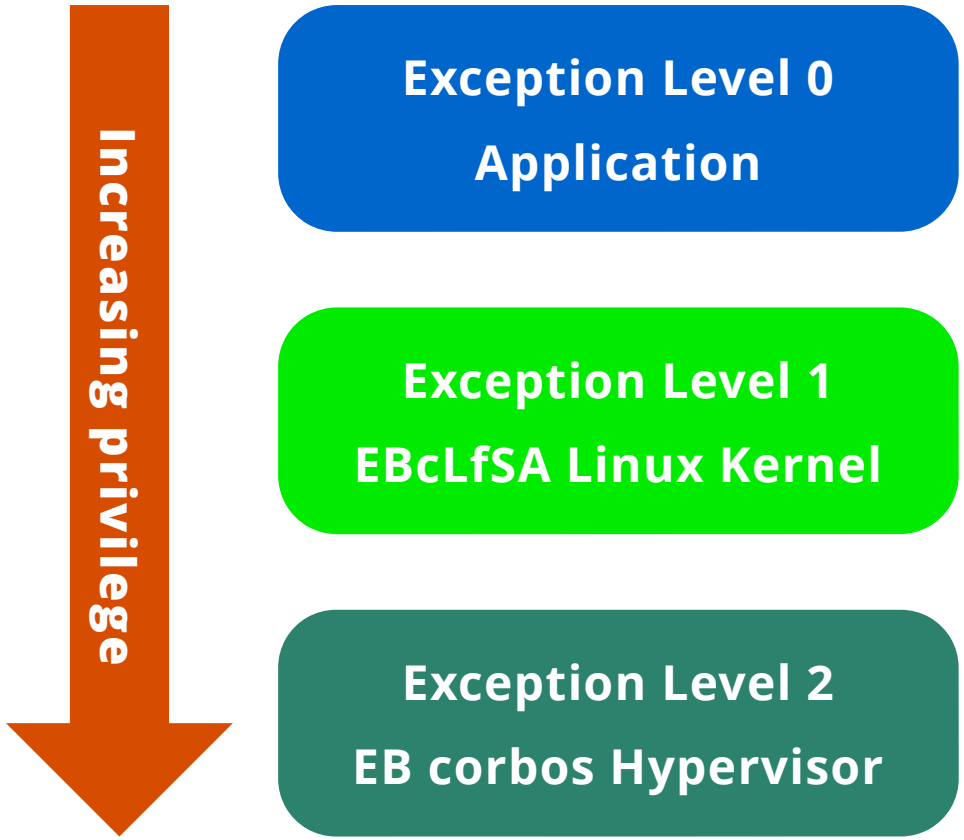
What does the EBcLfSA look like



To each their own: separation of virtual resources



How to get it: ARM AArch 64 Exception Levels



Exception Levels define the access rights

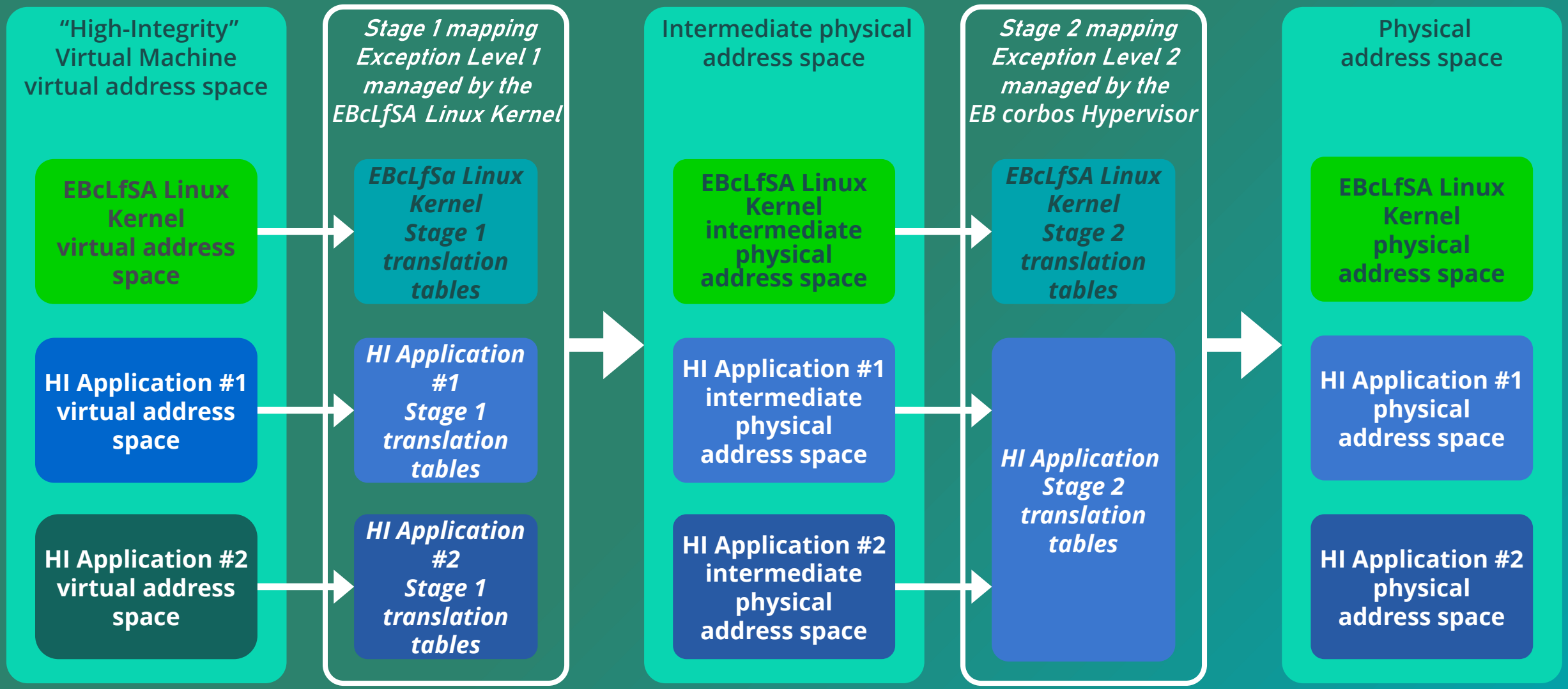
Both software elements and resources (memory, registers...) have Exception Levels

A software element can access only resources having the same or lower exception level

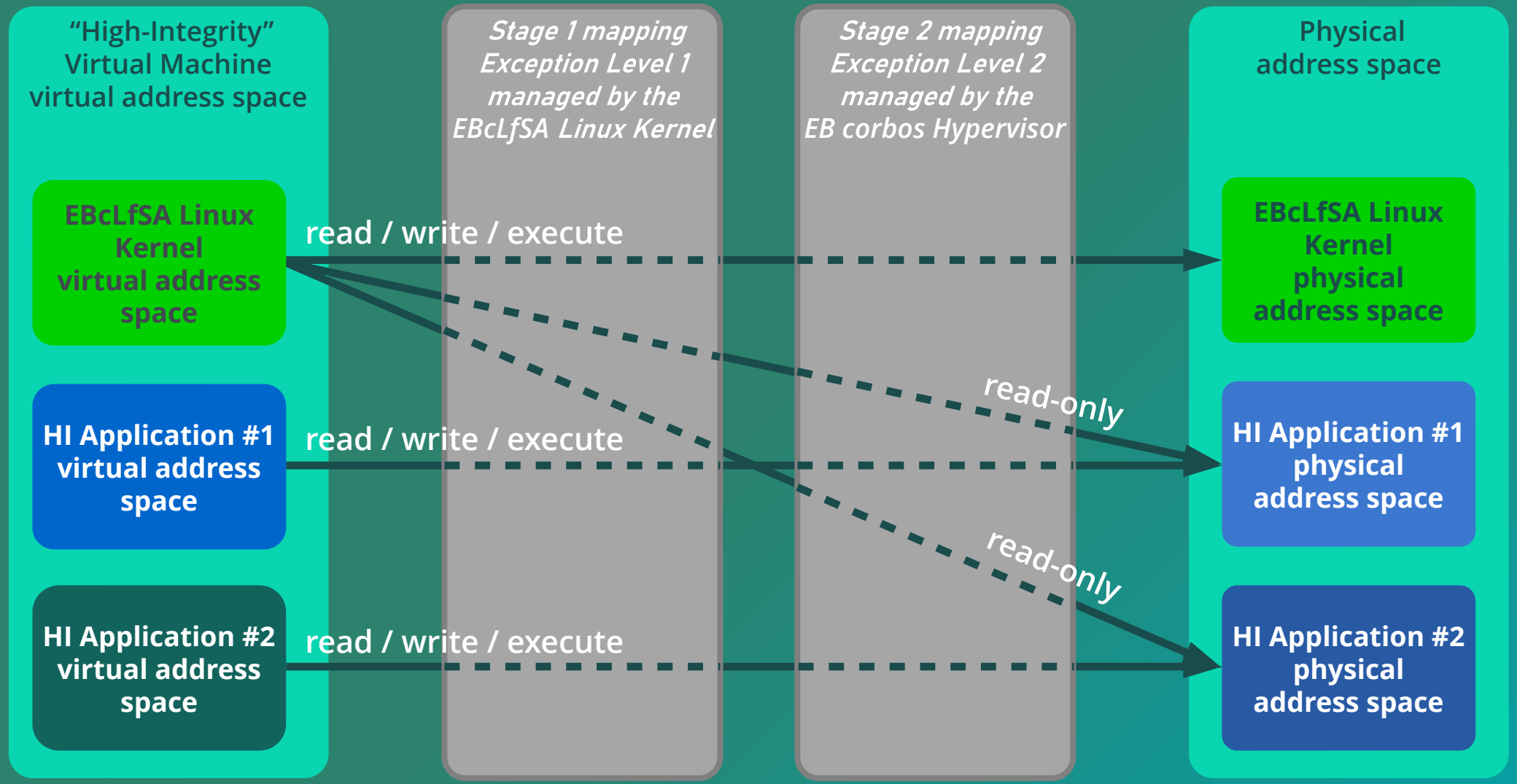
For instance, the EBcLfSA Linux Kernel runs at Exception Level 1:

- **it can access resources having Exception Level 1 or Exception Level 0**
- **it cannot access resources having Exception Level 2 or Exception Level 3**

How to use it: multi-stage virtualisation (ARM AArch 64)



The result: differentiated access rights (ARM AArch 64)



How does the separation work?

Access to the physical memory happens through the Stage 2 translation tables (Exception Level 2) which are managed by the EB corbos Hypervisor

The EB corbos Hypervisor knows “who” is trying to access “what” memory

The separation is that the EB corbos Hypervisor does not allow the EBcLfSA Linux Kernel to write or execute the memory of a HI Application

If the EBcLfSA tries to write or execute the memory of a HI Application:

- the EB corbos Hypervisor detects a violation and invokes the Supervisor as the exception handler
- the Supervisor analyses the access and
  - if the access is legitimate, it actually executes it dependably
  - if the access is not legitimate, it blocks it and does not allow it

And what does that mean for the HI Application?

The dependability of the HI Application consists of the following:

- only the HI Application itself can write or execute the memory allocated to it
- any write or execute to the memory allocated to a HI Application is supervised
- any legitimate access by the EBcLfSA Linux Kernel to the memory allocated to a HI Application is actually (and dependably) executed by the EB corbos Hypervisor and the Supervisor

But also:

- when the HI Application is resumed following a context switch, the Supervisor checks that the system context is correctly restored
- the EB corbos Hypervisor checks that the HI Application is actually being executed

Under these conditions, the HI Application is dependable

And now the bad news: what's NOT in the package

The functional correctness of the EBcLfSA Linux Kernel is NOT ensured:

- it is ensured that the Linux does not adversely affect the dependability of the HI Application as described, but it is NOT ensured that it does what it is expected to do

The correctness of data supplied to the HI Application is NOT ensured:

- no dependability claim is made on that data

Detection, mitigation and negation of credible hardware faults is NOT provided

- the matter is considered project-specific

The EBcLfSA detects violations of the dependability of the HI Application, but does NOT implement the mitigation or negation:

- mitigation and negation of detected issues is considered project-specific and needs to be defined and implemented at a higher level of integration

In summary: who does what and what it gets

Features provided by the hardware (ARM AArch64 architecture)

- exception levels
- multi-stage virtualization
- Memory Management Unit

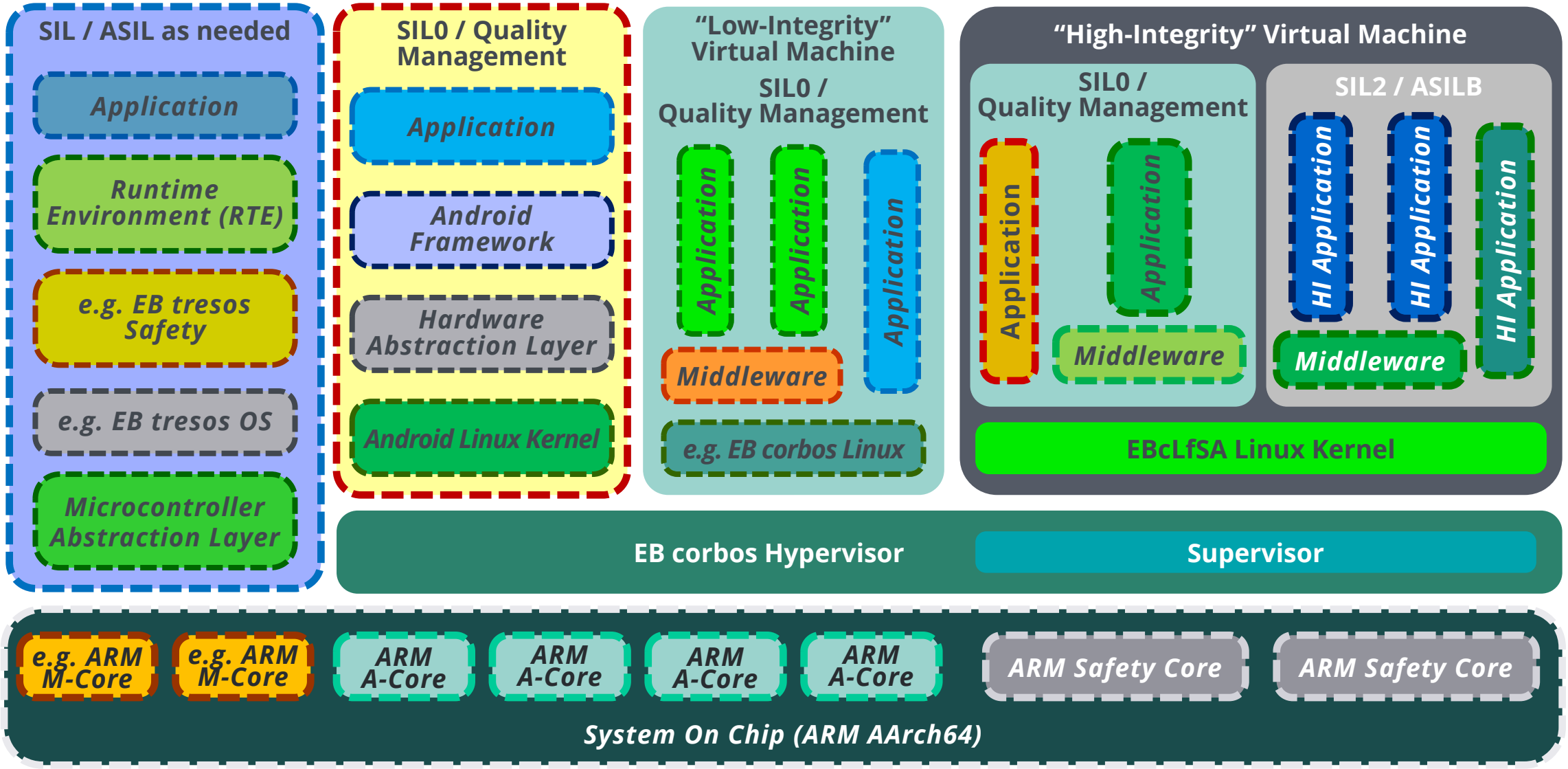
Features provided by the EB corbos Hypervisor

- separation of virtual address space for the “High-Integrity” Virtual Machine
- separation of the virtual address spaces between the EBcLfSA Linux Kernel and the HI Application
- trapping attempted accesses to memory or registers
- invoking the Supervisor
- checking that the HI Application is actually being executed
- confirming that the HI Application is supervised

Resulting features implemented in the EBcLfSA

- correct loading of memory segments
- protection of the memory of the HI Application
- correct restoring of the execution context for the HI Application
- detection of violations of the above points
- providing information about the health status

But wait: there's more!



Thank you for your attention

Federico Arrighetti

federico.arrighetti@elektrobit.com

Elektrobit Automotive GmbH
www.elektrobit.com

Michael Armbruster

solutions@emlix.com

emlix GmbH
www.emlix.com

21 May 2025